Thesis/
Reports
Bevins,
C. D.

THE STICK FUEL MOISTURE MODEL

Mr. Collin Bevins

33 Bookplate
(1—63)

NATIONAL

AGRICULTURAL

LIBRARY

RMRS Library - Ogden

Library - Ogden Center
Rocky Mountain Research Station
324 25th Street
Ogden, UT 84401

FINAL REPORT FOR RESEARCH AGREEMENT
#INT-97066-RJVA
"Stick Fuel Moisture Model"
SYSTEMS FOR ENVIRONMENTAL MANAGEMENT
FS Contact: Ms. Patricia Andrews
CoOp Contact: Mr. Collin Bevins

The Stick Fuel Moisture Model

Research Joint Venture Agreement INT-97066-RJVA
between the
U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station
and
Systems for Environmental Management

Final Report
Collin D. Bevins
October 31, 1997

Introduction and Purpose

This is the final report to Research Joint Venture Agreement INT-97066-RJVA between the U.S. Department of Agriculture, Forest Service, Rocky Mountain Research Station, and System for Environmental Management.

The purpose of the agreement is to develop, test, refine, and document a computer model for predicting the fuel moisture content of 1/2-inch sticks using the physically-based theoretical model under development by Dr. Ralph Nelson of the Forest Service.

Tasks, Products, and Accomplishments

The following tasks were accomplished under this agreement.

1. Ralph Nelson's original, experimental FORTRAN 66 code was totally rewritten into a modular, structured FORTRAN 77 program suitable for further development and testing. The original source code was slowly evolved over 47 program generations. Each generation was tested using the three data sets to ensure consistency in output values.

2. A one-week collaboration and training session was conducted with Dr. Nelson and Collin Bevins to resolve the final design and implementation of the program, *stick47.f77*. During this period Dr. Nelson was trained in the editing, compilation, and application of the program in furtherance of theoretical model development and testing.

3. Over the next several weeks Dr. Nelson implemented numerous changes in the model and returned a copy of the new (*stick48.f77*) code to Collin Bevins. Changes include improvements to the moisture diffusivity computations, a greatly reduced simulation time step increment, and new values for some parameterized constants.

4. Collin Bevins optimized the final FORTRAN 77 version of the model (*stick49.f77*). Both the non-optimized and the optimized versions were run with all three test data sets to verify agreement in the outputs.

5. A C function application programming interface, the Fuel Moisture Stick Function Library, was then designed and developed from the *stick49.f77* FORTRAN code. The C API allows the stick fuel moisture model to be incorporated in any number of fire, fuel, and moisture applications. The API employs an object-oriented design, allowing an application to create, modify, update, and destroy any number of fuel stick instances during a simulation.

6. A C program, *fmstest*, was developed to exercise the FMS library and demonstrate its use. Again, all three test data sets were processed through *fmstest*, and outputs validated against those of the FORTRAN program. The time required to run the test data on two

different computer systems are:

Data Set	Number of Hourly Observations	Pentium 100	RS/6000 590
Burnsville, N.C	304	2 sec	<1 sec
Mio, MI	301	3 sec	< 1 sec
Missoula, MT	1121	10 sec	3 sec

The table illustrates that model predictions can be made for a year of hourly observations (8760 observations) in about 75 seconds on a fairly outdated P100 laptop computer.

7. A Tcl command interface was created for the FMS Library. This C function library permits fuel moisture stick objects to be created, manipulated, updated, and destroyed from within the Tcl scripting language. A short, 137 line Tcl/Tk program, *fmstick.tcl*, was developed to demonstrate the Tcl/FMS scripting capability. It reads a test data set and displays a graph of observed vs. predicted values on the terminal screen.

8. The FMS Library, *fmstest* program, FMS Tcl/Tk interface, *fmstick.tcl* demonstration program, and the three test data sets were installed, compiled, and tested on (1) an Intel P100 running UnixWare 2.01, and (2) an IBM RS/6000 590 running AIX 4.1. The FMS library and *fmstest* program were also compiled and tested on MS-DOS/Win95 operating systems.

9. A paper titled *The Fuel Moisture Stick Function Library* was written documenting all FMS library functions and the *fmstest* C program and usage. Appendix A devotes a separate page to each function call including its name, synopsis, arguments, return values, and description. Appendix B lists the FMS Library source code files *fms.c* and *fms.h*, while Appendix C lists the *fmstest.c* program.

Acknowledgements

This research was supported in part by funds provided by the Rocky Mountain Research Station, Forest Service, U.S. Department of Agriculture.

This research was supported in part by funds provided by Systems for Environmenatl Management, a Montana non-profit research and educational organization.

The Fuel Moisture Stick Function Library

Collin D. Bevins
Systems for Environmental Management

Introduction

The Fuel Moisture Stick (FMS) library is a small collection of ANSI C functions implementing Nelson's (in preparation) 10-hour stick fuel moisture prediction model. The model estimates moisture content within standard 1/2-inch fuel sticks given periodic inputs of air temperature, air humidity, solar radiation, and cumulative rainfall.

The FMS library implements object-oriented programming constructs using the ANSI standard C programming language. The primary object is a "fuel moisture stick". Use of an object-oriented approach allows the creation, manipulation, and destruction of any number of fuel moisture stick instances as may be required by an application.

The FMS library contains 6 public (user callable) functions to create, manipulate, and destroy fuel moisture stick objects:

- *Fms_Create*() creates a standard 1/2-inch fuel moisture stick object.
- *Fms_CreateCustom*() creates a fuel moisture stick object with custom radius, length, density, and other properties.
- *Fms_Destroy*() destroys a fuel moisture stick object and frees all computing resources associated with it.
- *Fms_Initialize*() initializes the internal and environmental state of a new fuel moisture stick object.
- *Fms_Update*() calculates the stick's nodal moisture content values given current air temperature, air humidity, solar radiation, cumulative rainfall, and the elapsed time since the previous observation. Moisture content, diffusivity, and saturation point are calculated at evenly spaced radial nodes extending from the stick center to its outer surface.
- *Fms_MeanMoisture*() calculates the mean moisture content of a fuel moisture stick object using a simple integration algorithm.

In addition the FMS library contains 3 private functions used internally by the public functions:

- *Fms_Diffusivity*() calculates moisture diffusivity at each node within the fuel moisture stick object. It is called by *Fms_Initialize*() and *Fms_Update*().

- *Fms_Reconfigure*() reallocates computer resources for a fuel moisture stick object. It is called by *Fms_CreateCustom*().

- *Fms_Rederive*() sets manifest constants and calculates invariant intermediate factors used in *Fms_Update*(). Isolating the assignment and computation of these constants in a separate function permits comparison testing of different parameter sets between fuel moisture stick objects. It is called by *Fms_CreateCustom*().

Function Usage

Appendix A contains the FMS Library Function Reference, documenting each function's name, usage, arguments, return values, and description.

Appendix B lists the **fms.c** C source and **fms.h** C header files. The source code is heavily commented, documenting the names, descriptions, and units of all variables.

The normal calling sequence of FMS functions is:

1. Call *Fms_Create*() or *Fms_CreateCustom*() to create a fuel moisture stick instance.

2. Call *Fms_Initialize*() to initialize the stick's internal and environmental state.

3. For each weather observation:

    4. Call *Fms_Update*() to update the stick's nodal moisture contents.

    5. Call *Fms_MeanMoisture*() to get stick mean moisture content.

6. Call *Fms_Destroy*() to destroy the stick instance.

Example Program: fmstest

An example program, **fmstest**, has been developed to demonstrate FMS library usage. **fmstest** uses command line arguments to create and initialize a single fuel moisture stick instance. It then reads a set of weather observations, updating and printing the predicted 10-hour fuel stick moisture contents.

Table 1 lists the command line arguments accepted by **fmstest**. Appendix C lists the **fmstest.c** source code file. The first 3 pages of the source code file more thoroughly document the program usage.

fmstest Command Line Options		
Option	Description	Default
	Initial Stick Environment Options	
-hair	Initial air relative humidity.	0.80 g/g
-hfuel	Initial stick surface relative humidity.	0.80 g/g
-moisture	Initial stick moisture content.	0.20 g/g
-rain	Initial cumulative rainfall.	0.0 cm
-solar	Initial solar radiation.	0 watts/m2
-tair	Initial air temperature.	10.0 C
-tfuel	Initial fuel stick temperature.	10.0 C
	Fuel Stick Property Options	
-ddt	Moisture diffusivity calculation time interval.	0.25 hours
-density	Stick density.	0.40 g/cm3
-length	Stick length.	50.0 cm
-mdt	Moisture content calculation time interval.	1/60 hour
-nodes	Number of moisture content computation nodes.	21
-pressure	Barometric pressure.	.0218 cal/cm3
-radius	Stick radius.	0.64 cm
-wmax	Maximum stick moisture content.	0.60 g/g
	Input - Output Options	
-annotate	Add annotations to output? ("yes" or "no").	no
-delimit	Use "space" or "comma" for output field delimiters.	space
-input	Input file name.	stdin
-output	Output file name.	stdout
-verbose	Include weather variables in output? ("yes" or "no").	no

fmstest was used during development of the FMS library to test function adequacy and performance and to generate predicted vs observed moisture content pairs. Table 2 lists program run times for three data sets on two computing platforms.

fmstest Performance Results			
Data Set	Hourly Observations	Pentium 100 (seconds)	RS/6000 590 (seconds)
Burnsville, NC	304	2	< 1
Mio, MI	301	3	< 1
Missoula, MT	1121	10	3

fmstest is built using the following compiler commands:

```
cc -O fmstest.c fms.c -lm -o fmstest
```

Acknowledgement

This research was supported in part by funds provided by the Rocky Mountain Research Station, Forest Service, U.S. Department of Agriculture.

This research was supported in part by funds provided by Systems for Environmental Management, a Montana non-profit research and educational organization.

Appendix A

Fuel Moisture Stick Library Reference Manual

NAME

Fms_Create()

SYNOPSIS

```
#include "fms.h"
Fms *Fms_Create ( char *name ) ;
```

ARGUMENTS

name Name of the new fuel moisture stick object.

RETURNS

Pointer to the new Fms object on success or NULL on failure.

DESCRIPTION

Creates a "standard" 10-hour pine fuel moisture stick object.

NAME

Fms_CreateCustom()

SYNOPSIS

```
#include "fms.h"
Fms *Fms_CreateCustom ( char *name, size_t nodes, double radius, double density,
        double mdt, double ddt, double pressure, double wmx ) ;
```

ARGUMENTS

name	Name of the new fuel moisture stick object.
nodes	Number of calculation nodes.
radius	Stick radius (cm).
length	Stick length (cm).
density	Stick density (gm/cm3).
mdt	Nodal moisture content computation time step (hr).
ddt	Moisture diffusivity computation time step (hr).
pressure	Barometric pressure (cal/cm3).
wmx	Maximum local moisture content due to rain (g/g).

RETURNS

Pointer to the new Fms object on success or NULL on failure.

DESCRIPTION

Creates a "custom" 10-hour pine fuel moisture stick object.

NOTES

Manifest constants used by the fuel moisture computations are assigned to each object. This permits experimentation and comparison between sticks using different "constants". Fms_CreateCustom() calls the static Fms_CreateConstants() to compute "derived constants" and other optimization factors that are based solely on the "manifest" constants.

A program wishing to change or test the constants should:

(1) Call Fms_CreateCustom(...),

(2) Directly change any constants for the Fms object; for example:

```
fms->Alb = 0.55
```

(3) Call Fms_Rederive(fms)

(4) Call Fms_Reconfigure(fms)

NAME

Fms_Destroy()

SYNOPSIS

```
#include "fms.h"
void Fms_Destroy ( Fms *fms ) ;
```

ARGUMENTS

fms Pointer to the Fms object to be destroyed.

RETURNS

Returns nothing.

DESCRIPTION

Deletes the fuel moisture stick object and free all its resources.

NAME

Fms_Initialize()

SYNOPSIS

```
#include "fms.h"
void Fms_Initialize ( Fms *fms, double ta, double ha double sv, double ti, double wi );
```

ARGUMENTS

fms	Pointer to the fuel moisture stick object.
ta	Initial air temperature (C).
ha	Initial air humidity (g/g).
sv	Initial solar voltage (millivolts).
ti	Initial stick temperature (C).
hi	Initial stick surface humidty (g/g).
wi	Initial stick water fraction (/g).

RETURNS

Function returns nothing.

DESCRIPTION

Initializes the stick's internal and environmental variables. The stick's internal temperature and water content are presumed to be uniformly distributed.

NAME

Fms_MeanMoisture()

SYNOPSIS

```
#include "fms.h"
double Fms_MeanMoisture ( Fms *fms ) ;
```

ARGUMENTS

fms Pointer to fuel moisture stick object.

RETURNS

Fuel moisture stick's mean moisture content (g/g).

DESCRIPTION

Calculates the stick's mean moisture content over all nodes.

NAME

Fms_Update()

SYNOPSIS

```
#include "fms.h"
void Fms_Update ( Fms *fms, double eHr, double aC, double hFtn, double sMv,
        double rcumCm ) ;
```

ARGUMENTS

fms Pointer to fuel moisture stick object to update.

RETURNS

The function returns nothing.

DESCRIPTION

Ralph Nelson's 10-hour stick fuel moisture prediction model. Updates the stick's temperature and moisture variables based upon changes in the stick's environmental variables since the last update.

SIDE EFFECTS

Updates the sticks temperature (fms->t), saturation (fms->s[]), and moisture (fms->w[]) values. If the diffusivity time limit is exceeded, diffusivity (fms->d) is also updated.

NAME

Fms_Diffusivity()

SYNOPSIS

```
#include "fms.h"
static void Fms_Diffusivity ( Fms *fms ) ;
```

ARGUMENTS

fms Pointer to fuel moisture stick object.

RETURNS

The stick's nodal bound water diffusivity values fms->d[i] (cm2/hr) are recalculated.
The function returns nothing.

DESCRIPTION

Calculates the stick's nodal bound water diffusivity (cm2/hr).

NAME

Fms_Reconfigure()

SYNOPSIS

```
#include "fms.h"
static long Fms_Reconfigure ( Fms *fms ) ;
```

ARGUMENTS

fms Pointer to fuel moisture stick object.

RETURNS

Returns FMS_OK on success or FMS_ERROR on failure.

DESCRIPTION

Recalculates all stick elements from current node, radius, density, time step, and barometric pressure values. Also calculates intermediate parameters and optimization factors that depend solely on manifest constants, derived constants, and stick properties.

NAME

Fms_Rederive()

SYNOPSIS

```
#include "fms.h"
Fms *Fms_Rederive ( Fms *fms ) ;
```

ARGUMENTS

fms Pointer to the Fms object to derive constants.

RETURNS

Returns nothing.

DESCRIPTION

Derives all the "constants" and optimization factors that rely solely on the Fms object's "manifest constants".

NOTES

Manifest constants used by the fuel moisture computations are assigned to each object in Fms_CreateCustom(). This permits experimentation and comparison between sticks using different "constants". Derived constants and optimization factors based solely upon the manifest constants are then calculated here.

A program wishing to change or test the constants should:

(1) Call Fms_CreateCustom(...),

(2) Directly change any constants for the Fms object; for example:

```
fms->Alb = 0.55
```

(3) Call Fms_Rederive(fms)

(4) Call Fms_Reconfigure(fms)

Appendix B

Fuel Moisture Stick Library
Source Code Files

fms.c
fms.h

```
1  /*
2  ******************************************************************************
3  *
4  *  NAME
5  *     fms.h
6  *
7  *  DESCRIPTION
8  *     This header file describes the externally-visible facilities of the
9  *     Ralph Nelson's 10-hour stick fuel moisture prediction model.
10 *
11 *  AUTHOR
12 *     Copyright (C) 1997 by Collin D. Bevins.
13 *     See the file "license.terms" for information on usage and
14 *     redistribution of this file, and for a DISCLAIMER OF ALL WARRANTIES.
15 *
16 ******************************************************************************
17 */
18
19 #ifndef _FMSTICK_LIB
20 #define _FMSTICK_LIB 1
21
22 #include <stdlib.h>
23 #include <string.h>
24 #include <math.h>
25
26 #ifdef __cplusplus
27 #   define EXTERN extern "C"
28 #else
29 #   define EXTERN extern
30 #endif
31
32 /*
33  *---------------------------------------------------------------------------
34  * FMS library return codes.
35  *---------------------------------------------------------------------------
36  */
37
38 #define FMS_OK                (0)
39 #define FMS_ERROR             (-1)
40
41 /*
42  *---------------------------------------------------------------------------
43  * Stick moisture states.
44  *---------------------------------------------------------------------------
45  */
46
47 #define FMS_STATE_None           (0)
48 #define FMS_STATE_Adsorption     (1)
49 #define FMS_STATE_Desorption     (2)
50 #define FMS_STATE_Condensation   (3)
51 #define FMS_STATE_Evaporation    (4)
52 #define FMS_STATE_Rainfall       (5)
53 #define FMS_STATE_Rainstorm      (6)
54 #define FMS_STATE_Stagnation     (7)
55 #define FMS_STATE_Error          (8)
56 #define FMS_STATE_condensation   (9)
57
58 /*
59  *---------------------------------------------------------------------------
60  *  StickData structure.
61  *---------------------------------------------------------------------------
62  */
63
64 typedef struct
65 {
66     /* Stick "manifest" constants set in fms_CreateCustom() */
```

```
67
68      double Aks;     /* Permeability of water saturated stick (cm2).  */
69      double Alb;     /* Shortwave albido (dl).  */
70      double Alpha;   /* Portion of cell length overlaping adjacent cells (dl).  */
71      double Ap;      /* Psychrometric constant (1/C).  */
72      double Aw;      /* Ratio of cell cavity to total cell width.  */
73      double Eps;     /* Longwave emissivity of stick surface (dl).  */
74      double Hc;      /* Planar heat transfer / Pi (cal/cm2-hr-C).  */
75      double Kelvin;  /* Celsius-to-Kelvin offset (C).  */
76      double Pi;      /* A well-rounded number (dl).  */
77      double Pr;      /* Prandtl number (dl).  */
78      double Sbc;     /* Stefan-Boltzmann constant (cal/cm2-s-K4).  */
79      double Sc;      /* Schmidt number (dl).  */
80      double St;      /* ???  */
81      double Stcaf;   /* Min surface mass transfer for adsorption ((cm3/cm2)/hr) a
-2 t high or low stick surface rh.  */
82      double Stcd;    /* Surface mass transfer for desorption ((cm3/cm2)/hr).  */
83      double Tcd;     /* Day time clear sky temperature (C).  */
84      double Tcn;     /* Night time clear sky temperature (C).  */
85      double Wl;      /* ???  */
86
87      /* Stick "derived" constants set in Fms_Redefine().  */
88
89      double Srf;     /* Factor to derive "sr", solar rad received (cal/cm2-hr).  */
90      double Wsf;     /* -log(1.0 - 0.99).  */
91      double Hrd;     /* Factor to derive daytime longwave radiative surface heat
-2 transfer.  */
92      double Hrn;     /* Factor to derive nighttime longwave radiative surface hea
-2 t transfer.  */
93      double Sir;     /* Saturation below which columns of liquid water no longer
-2 exist.  */
94      double Scr;     /* Saturation value at which liquid minuscus first enters th
-2 e tapered portion of wood cells.  */
95      double Vf;      /* Optimization factor.  */
96      double Hwf;     /* Optimization factor.  */
97
98      /* Stick physical properties passed to Fms_CreateCustom().  */
99
100     double a;       /* in : stick radius                                  (cm)  */
101     double al;      /* in : stick length                                  (cm)  */
102     double dp;      /* in : stick density                              (g/cm3)  */
103     double bp;      /* in : stick barometric pressure                  (cal/cm3)  */
104     double mdt;     /* in : stick moisture maximum time step              (hr)  */
105     double ddt;     /* in : stick diffuse maximum time step               (hr)  */
106     double wmx;     /* in : stick maximum local moisture due to rain      (g/g)  */
107
108     /* Stick environmental variables.  */
109
110     double ta0;     /* in : previous air temperature                      (C)  */
111     double ta1;     /* in : current air temperature                       (C)  */
112     double ha0;     /* in : previous air humidity                        (dl)  */
113     double ha1;     /* in : current air humidity                         (dl)  */
114     double sv0;     /* in : previous solar radiation                     (mv)  */
115     double sv1;     /* in : current solar radiation                      (mv)  */
116     double f0;      /* in : previous cumulative rainfall                  (cm)  */
117     double f1;      /* in : current cumulative rainfall                   (cm)  */
118     double fcum;    /* out: cumulative hourly rainfall rate            (cm/hr)  */
119     double rdur;    /* out: duration of current rainfall                  (hr)  */
120
121     /* Stick derived intermediate constants.  */
122
123     double dx;      /* out: internodal radial distance                    (cm)  */
124     double wmax;    /* max possible stick moisture content                (g/g)  */
125
126     /* Stick-dependent optimization factors.  */
127     double amlf;
```

```
128     double capf;
129     double pro;
130     double mdt_2;
131     double dx_2;
132     double sf;
133
134     /* Stick derived output parameters. */
135
136     double tf;      /* out: stick temperature                          (C) */
137     double hf;      /* out: stick surface humidity                   (g/g) */
138     double wsa;     /* out: stick fiber saturation point             (g/g) */
139     double sem;     /* out: stick equilibrium moisture content       (g/g) */
140     double wfilm;   /* out: amount of water film on stick surface    (g/g) */
141     double *x;      /* out: node's radial distance from center of stick (cm) */
142     double *s;      /* out: node's saturation                        (g/g) */
143     double *d;      /* out: node's bound water diffusivity         (cm2/hr) */
144     double *w;      /* out: node's moisture content                  (g/g) */
145
146     char  *name;    /* in : stick name                                     */
147     long   state;   /* out: FMS_STATE_<state>                              */
148     long   updates; /* out: calls to Fms_Update() (e.g., num of weather obs) */
149     long   init;    /* out: cleared by Fms_Reconfigure, set by Fms_Initialize */
150     size_t n;       /* in : nodes in radial direction                  (dl) */
151 } Fms;
152
153 /*
154  *-------------------------------------------------------------------------
155  * Public Function prototypes.
156  *-------------------------------------------------------------------------
157  */
158
159 EXTERN Fms *Fms_Create ( char * name ) ;
160
161 EXTERN Fms *Fms_CreateCustom (
162     char  *name,    /* Name for the new stick instance.                    */
163     size_t nodes,   /* Number of calculation nodes.                        */
164     double radius,  /* Stick radius                                   (cm) */
165     double length,  /* Stick length                                   (cm) */
166     double density, /* Stick density                               (gm/cm3) */
167     double mdt,     /* Moisture computation time step                 (hr) */
168     double ddt,     /* Diffusivity computation time step              (hr) */
169     double pressure,/* Barometric pressure                         (cal/cm3) */
170     double wmx      /* Max local moisture due to rain                (g/g) */
171 ) ;
172
173 EXTERN void Fms_Destroy ( Fms *fms ) ;
174
175 EXTERN void Fms_Initialize (
176     Fms    *fms,    /* Pointer to fuel moisture stick object.              */
177     double  ta,     /* Initial air temperature                        (C) */
178     double  ha,     /* Initial air humidity                         (g/g  */
179     double  sv,     /* Initial solar voltage                         (mV) */
180     double  rc,     /* Initial cumulative rainfall                   (cm) */
181     double  ti,     /* Initial stick temperature                      (C) */
182     double  hi,     /* Initial stick surface humidity              (g/g) */
183     double  wi      /* Initial stick moisture content              (g/g) */
184 ) ;
185
186 EXTERN double Fms_MeanMoisture ( Fms *fms ) ;
187
188 EXTERN void Fms_Rederive ( Fms *fms ) ;
189
190 EXTERN void Fms_Update (
191     Fms   *fms,     /* Pointer to the fuel moisture stick object.          */
192     double eHr,     /* Elapsed time since last observation           (hr) */
193     double aC,      /* New air temperature                            (C) */
```

```
194     double hFtn,                    /* New air humidity                 (g/g) */
195     double sMv,                     /* New solar radiation          (milliVolts) */
196     double rcumCm                   /* New cumulative rainfall             (cm) */
197 ) ;
198
199 #endif
200
201 /*
202  ************************************************************************************
203  *  End of fms.h
204  ************************************************************************************
205  */
```

```
/*
*******************************************************************************
*
*  NAME
*      fms.c
*
*  DESCRIPTION
*      Ralph Nelson's 10-hour stick fuel moisture prediction model.
*
*  AUTHOR
*      Copyright (c) 1997 by Collin D. Bevins.
*      See the file "license.terms" for information on usage and
*      redistribution of this file, and for a DISCLAIMER OF ALL WARRANTIES.
*
*  PUBLIC FUNCTIONS
*      Function            Usage
*      ----------------    --------
*      Fms_Create          Normal
*      Fms_CreateCustom    Advanced
*      Fms_Destroy         Normal
*      Fms_Initialize      Normal
*      Fms_MeanMoisture    Normal
*      Fms_Rederive        Advanced
*      Fms_Update          Normal
*
*  PRIVATE FUNCTIONS
*      Fms_Diffusivity
*      Fms_Reconfigure
*
*  NORMAL USAGE
*      Four or five functions are normally used:
*      1 Call Fms_Create() to create a fuel moisture stick object.
*      2 Call Fms_Initialize() to initialize the stick's environment and
*          internal state.
*      3 Call Fms_Update() within a loop to update the stick's fuel
*          temperature, humidity, and moisture content for a set of
*          weather observations
*      4 Call Fms_MeanMoisture() whenever stick's mean moisture content
*          is required.
*      5 Optionally call Fms_Destroy() to destroy the stick instance.
*
*  ADVANCED USAGE
*      1 Call Fms_CreateCustom() to create a stick with non-standard radius,
*          length, density, maximum moisture content, moisture comuptation
*          time step, diffusivity computation time step, barometric
*          pressure, and number of nodes.
*      2 Call Fms_Rederive() after modifying any of the stick computation
*          constants to update derived constants.
*
*  HISTORY
*      Ver 1.0 : Created from stick47.for migration path.
*******************************************************************************
*/

#include "fms.h"

static void Fms_Diffusivity ( Fms *fms ) ;
static long Fms_Reconfigure ( Fms *fms ) ;

/*
*-------------------------------------------------------------------------------
*  Base manifest constants.
*-------------------------------------------------------------------------------
*/

#define FMS_MAX_NODES    (128)        /* Used for nodal automatics only! */
```

```
67
68  /*
69   ***************************************************************************
70   *
71   *  NAME
72   *      Fms_Create()
73   *
74   *  SYNOPSIS
75   *      #include "fms.h"
76   *      Fms *Fms_Create ( char *name );
77   *
78   *  ARGUMENTS
79   *      name    Name of the new fuel moisture stick object.
80   *
81   *  RETURNS
82   *      Pointer to the new Fms object on success or NULL on failure.
83   *
84   *  DESCRIPTION
85   *      Creates a "standard" 10-hour pine fuel moisture stick object.
86   ***************************************************************************
87   */
88
89  Fms *Fms_Create ( char *name )
90  {
91      return ( Fms_CreateCustom (
92          name,                       /* stick label */
93          21,                         /* calculation nodes */
94          0.64,                       /* stick radius (cm) */
95          50.,                        /* stick length (cm) */
96          0.40,                       /* stick density (g/cm3) */
97          (1./60.),                   /* maximum moisture time step (hr) */
98          .25,                        /* maximum diffusivity time step (hr) */
99          0.0218,                     /* barometric pressure (cal/cm3) */
100         0.60)                       /* max moisture content from rain (g/g) */
101     );
102 }
103
104 /*
105  ***************************************************************************
106  *
107  *  NAME
108  *      Fms_CreateCustom()
109  *
110  *  SYNOPSIS
111  *      #include "fms.h"
112  *      Fms *Fms_CreateCustom ( char *name, size_t nodes, double radius,
113  *          double density, double mdt, double ddt, double pressure,
114  *          double wmx ) ;
115  *
116  *  ARGUMENTS
117  *      name        Name of the new fuel moisture stick object.
118  *      nodes       Number of calculation nodes.
119  *      radius      Stick radius (cm).
120  *      length      Stick length (cm).
121  *      density     Stick density (gm/cm3).
122  *      mdt         Moisture computation time step (hr).
123  *      ddt         Diffusivity computation time step (hr).
124  *      pressure    Barometric pressure (cal/cm3)
125  *      wmx         Maximum local moisture content due to rain (g/g).
126  *
127  *  RETURNS
128  *      Pointer to the new Fms object on success or NULL on failure.
129  *
130  *  DESCRIPTION
131  *      Creates a "custom" 10-hour pine fuel moisture stick object.
132  *
```

```
133  *  NOTES
134  *      Manifest constants used by the fuel moisture computations are assigned
135  *      to each object.  This permits experimentation and comparison between
136  *      sticks using different "constants".  Fms_CreateCustom() calls the
137  *      static Fms_CreateConstants() to compute "derived constants" and other
138  *      optimization factors that are based solely on the "manifest" constants.
139  *
140  *      A program wishing to change or test the constants should:
141  *          (1) Call Fms_CreateCustom(...);
142  *          (2) Directly change any constants for the Fms object, for example:
143  *                  fms->Alb = 0.55
144  *          (3) Call Fms_Rederive(fms)
145  *          (4) Call Fms_Reconfigure(fms)
146  ************************************************************************************
147  */
148
149  Fms *Fms_CreateCustom (
150      char  *name,                    /* Name for the new stick instance          */
151      size_t nodes,                   /* Number of calculation nodes.             */
152      double radius,                  /* Stick radius                      (cm)   */
153      double length,                  /* Stick length                      (cm)   */
154      double density,                 /* Stick density                  (gm/cm3)  */
155      double mdt,                     /* Moisture computation time step    (hr)   */
156      double ddt,                     /* Diffusivity computation time step (hr)   */
157      double pressure,                /* Barometric pressure            (cal/cm3)  */
158      double wmx                      /* Max local moisture due to rain    (g/g)  */
159  )
160  {
161      Fms *fms;
162
163      /* Allocate the basic fms object. */
164      if ( (fms = (Fms *) malloc(sizeof(Fms))) == NULL )
165          return(NULL);
166
167      /* Store the passed parameters. */
168      fms->name = (char *) malloc(strlen(name)+1);
169      strcpy(fms->name, name);
170      fms->n    = nodes;
171      fms->a    = radius;
172      fms->al   = length;
173      fms->dp   = density;
174      fms->bp   = pressure;
175      fms->mdt  = mdt;
176      fms->ddt  = ddt;
177      fms->wmx  = wmx;
178
179      /* Initialize some other parameters and nodal array pointers. */
180      fms->x    = NULL;
181      fms->d    = NULL;
182      fms->s    = NULL;
183      fms->w    = NULL;
184
185      /*
186       *-----------------------------------------------------------------------
187       * Assign "manifest constants".
188       *-----------------------------------------------------------------------
189       */
190
191      /* Aks = permeability of water saturated stick (cm2). */
192      fms->Aks = 2.0e-13;
193
194      /* Alb = shortwave albido (dl). */
195      fms->Alb = 0.6;
196
197      /* Alpha = fraction of cell length that overlaps adjacent cells (dl). */
198      fms->Alpha = 0.25;
```

```
199
200     /* Ap = psychrometric constant (1/C) . */
201     fms->Ap = 0.000772;
202
203     /* Aw = ratio of cell cavity to total cell width. */
204     fms->Aw = 0.8;
205
206     /* Eps = longwave emissivity of stick surface (dl). */
207     fms->Eps = 0.85;
208
209     /* Hc = planar heat transfer/a Pi (cal/cm2-hr-C) . */
210     fms->Hc = 0.45;
211
212     /* Kelvin = Celsius-to-Kelvin offset (C). */
213     fms->Kelvin = 273.2;
214
215     /* Pi = a well-rounded number (dl). */
216     fms->Pi = 3.1416;
217
218     /* Pr = Prandtl number (dl). */
219     fms->Pr = 0.7;
220
221     /* Sbc = Stefan-Boltzmann constant (cal/cm2-s-K4). */
222     fms->Sbc = 1.37e-12;
223
224     /* Sc = Schmidt number (dl). */
225     fms->Sc = 0.58;
226
227     /* St = ??? */
228     fms->St = 72.8;
229
230     /* Stcaf = minimum surface mass transfer for adsorption ((cm3/cm2)/hr) */
231     /* at high or low stick surface rh. */
232     fms->Stcaf = 0.05;
233
234     /* Stcd = surface mass transfer for desorption ((cm3/cm2)/hr) */
235     fms->Stcd = 25.0;
236
237     /* Tcd = day time clear sky temperature (C). */
238     fms->Tcd = 5.;
239
240     /* Tcn = night time clear sky temperature (C). */
241     fms->Tcn = 0.;
242
243     /* W1 = ??? */
244     fms->W1 = 0.0023;
245
246     /* Calculate derived constants and optimization factors. */
247     Fms_Rederive(fms);
248
249     /* Call Fms_Reconfigure() to do the rest of the work. */
250     if ( Fms_Reconfigure(fms) != FMS_ERROR )
251         return (fms);
252     return (NULL);
253 }
254
255 /*
256  ***************************************************************************
257  *
258  *  NAME
259  *      Fms_Destroy()
260  *
261  *  SYNOPSIS
262  *      #include "fms.h"
263  *      void Fms_Destroy ( Fms *fms ) ;
264  *
```

```
265  *  ARGUMENTS
266  *      fms     Pointer to the Fms object to be destroyed.
267  *
268  *  RETURNS
269  *      Returns nothing.
270  *
271  *  DESCRIPTION
272  *      Deletes the fuel moisture stick object and free all its resources.
273  ***************************************************************************
274  */
275
276  void Fms_Destroy ( Fms *fms )
277  {
278      if ( fms == NULL ) return;
279      if ( fms->name ) free(fms->name);
280      if ( fms->x ) free(fms->x);
281      if ( fms->d ) free(fms->d);
282      if ( fms->s ) free(fms->s);
283      if ( fms->w ) free(fms->w);
284      free(fms);
285      return;
286  }
287
288  /*
289   ***************************************************************************
290   *
291   *  NAME
292   *      Fms_Diffusivity()
293   *
294   *  SYNOPSIS
295   *      #include "fms.h"
296   *      static void Fms_Diffusivity ( Fms *fms ) ;
297   *
298   *  ARGUMENTS
299   *      fms     Pointer to fuel moisture stick object.
300   *
301   *  RETURNS
302   *      The stick's nodal bound water diffusivity values fms->d[i] (cm2/hr)
303   *      are recalculated.
304   *      The function returns nothing.
305   *
306   *  DESCRIPTION
307   *      Calculates the stick's nodal bound water diffusivity (cm2/hr).
308   ***************************************************************************
309   */
310
311  static void Fms_Diffusivity ( Fms *fms )
312  {
313      size_t i;
314      double c1;      /* Emc sorption isotherm parameter                  (g/g) */
315      double c2;      /* Emc sorption isotherm parameter                  (g/g) */
316      double con;     /* Correction for tortuous paths in cell wall             */
317      double cpv;     /* Specific heat of water vapor          (cal/(mol*K)) */
318      double daw;     /* Density of adsorbed water                    (g/cm3) */
319      double dhdm;    /* Reciprocal slope of the sorption isotherm              */
320      double dhdm0;   /* Reciprocal slope of the sorption isotherm for stick rh */
321      double dhdm1;   /* Reciprocal slope of the sorption isotherm for 99% rh   */
322      double dv;
323      double dvpr;
324      double dvprf;   /* Intermediate factor for calculating dvpr               */
325      double e;       /* Activation energy for bound water diffusion  (cal/mol) */
326      double fac;     /* Converts D from wood substance to whole wood basis     */
327      double ps1;     /* Water satur vapor press at surface temp      (cal/cm3) */
328      double qv;      /* Latent heat of vaporization of water         (cal/mol) */
329      double qw;      /* Differential heat of sorption ofwater        (cal/mol) */
330      double rfcw;    /* Converts D from wood substance to whole wood basis     */
```

```
331     double svaw;     /* Specific volume of adsorbed water              (cm3/g) */
332     double tk;       /* Stick temperature                                (K) */
333     double vfaw;     /* Volume fraction of adborbed water               (dl) */
334     double vfcw;     /* Volume fraction of moist cell wall              (dl) */
335     double wc;       /* Lesser of nodal or fiber saturation moisture    (g/g) */
336
337     /* Hfs = saturation value of the stick surface relative humidity (g/g). */
338     static const double Hfs = 0.99;
339
340     tk    = fms->tf + 273.2;
341     qv    = 13550. - 10.22 * tk;
342     cpv   = 7.22 + .002374 * tk + 2.67e-07 * tk * tk;
343     dv    = 0.22 * 3600. * (0.0242 / fms->bp) * pow((tk / 273.2), 1.75);
344     ps1   = 0.0000239 * exp(20.58 - (5205. / tk));
345     c1    = 0.1617 - 0.001419 * fms->tf;
346     c2    = 0.4657 + 0.003578 * fms->tf;
347     dhdm0 = (1.0 - fms->hf) * pow(-log(1.0-fms->hf), (1.0 - c2)) / (c1 * c2);
348     dhdm1 = (1.0 - Hfs) * pow(fms->Wsf, (1.0 - c2)) / (c1 * c2);
349     dvprf = 18.0 * 0.016 * dv * ps1 / (fms->dp * 1.987 * tk);
350
351     for ( i=0; i<fms->n; i++ )
352     {
353         wc   = (fms->w[i] < fms->wsa) ? fms->w[i] : fms->wsa;
354         dhdm = (fms->w[i] < fms->wsa) ? dhdm0 : dhdm1;
355         daw  = 1.3 - 0.64 * wc;
356         svaw = 1. / daw;
357         vfaw = svaw * wc / (0.685 + svaw * wc);
358         vfcw = (0.685 + svaw * wc) / ((1.0 / fms->dp) + svaw * wc);
359         rfcw = 1.0 - sqrt(1.0 - vfcw);
360         fac  = 1.0 / (rfcw * vfcw);
361         con  = 1.0 / (2.0 - vfaw);
362         qw   = 5040. * exp(-14.0 * wc);
363         e    = (qv + qw - cpv * tk) / 1.2;
364
365         /*
366          * The factor 0.016 below is a correction for hindered water vapor diffusion
367          * (this diffusion is 62.5 times smaller than the bulk vapor diffusion);
368          * 0.0242 cal/cm3 = sea level atm pressure
369          *      -- Ralph Nelson
370          */
371
372 /*      dvpr = 18.0*0.016*(1.0-vfcw)*dv*ps1*dhdm / (fms->dp * 1.987 * tk); */
373         dvpr = dvprf * (1.0 - vfcw) * dhdm;
374         fms->d[i] = dvpr + 3600. * 0.0985 * con * fac * exp(-e/(1.987*tk));
375     }
376     return;
377 }
378
379 /*
380  ***************************************************************************
381  *
382  *  NAME
383  *      Fms_Initialize()
384  *
385  *  SYNOPSIS
386  *      #include "fms.h"
387  *      void Fms_Initialize ( Fms *fms, double ta, double ha double sv,
388  *          double ti, double wi );
389  *
390  *  ARGUMENTS
391  *      fms     Pointer to the fuel moisture stick object.
392  *      ta      Initial air temperature (C).
393  *      ha      Initial air humidity (g/g).
394  *      sv      Initial solar voltage (millivolts).
395  *      ti      Initial stick temperature (C).
396  *      hi      Initial stick surface humidty (g/g).
```

```
 *      wi      Initial stick water fraction (f/g).
 *
 *  RETURNS
 *      Function returns nothing.
 *
 *  DESCRIPTION
 *      Initializes the stick's internal and environmental variables.
 *      The stick's internal temperature and water content are presumed to be
 *      uniformly distributed.
 ************************************************************************
 */

void Fms_Initialize (
    Fms    *fms,                /* Pointer to fuel moisture stick object.     */
    double  ta,                 /* Initial air temperature                (C) */
    double  ha,                 /* Initial air humidity                 (g/g) */
    double  sv,                 /* Initial solar voltage                 (mV) */
    double  rc,                 /* Initial cumulative rainfall           (cm) */
    double  ti,                 /* Initial stick temperature              (C) */
    double  hi,                 /* Initial stick surface humidity       (g/g) */
    double  wi                  /* Initial stick moisture content       (g/g) */
)
{
    size_t i;
    double diffusivity;

    /* Environment initialization. */
    fms->ta0 = fms->ta1 = ta;
    fms->ha0 = fms->ha1 = ha;
    fms->sv0 = fms->sv1 = sv;
    fms->f0  = fms->f1  = rc / fms->Pi;
    fms->fcum = 0.;
    fms->rdur = 0.;

    /* Stick initialization. */
    fms->tf = ti;
    fms->hf = hi;
    for (i=0; i<fms->n; i++)
    {
        fms->w[i] = wi;
        fms->s[i] = 0.;
    }
    fms->wsa = wi + .1;
    Fms_Diffusivity(fms);
    fms->wfilm = 0;
    fms->init = 1;
    return;
}

/*
 ************************************************************************
 *
 *  NAME
 *      Fms_MeanMoisture()
 *
 *  SYNOPSIS
 *      #include "fms.h"
 *      double Fms_MeanMoisture ( Fms *fms );
 *
 *  ARGUMENTS
 *      fms     Pointer to fuel moisture stick object.
 *
 *  RETURNS
 *      Fuel moisture stick's mean moisture content (g/g).
 *
 *  DESCRIPTION
```

```
463  *      Calculates the stick's mean moisture content over all nodes.
464  ***************************************************************************
465  */
466
467  double Fms_MeanMoisture ( Fms *fms )
468  {
469      size_t i;
470      double wea, web, wec, wei, wbr;
471
472      wec = fms->w[0];
473      wei = fms->dx / (3. * fms->a);
474
475      for (i=1; i<fms->n-1; i+=2)
476      {
477          wea = 4. * fms->w[i];
478          web = 2. * fms->w[i+1];
479          if ((i+1) == (fms->n-1))
480              web = fms->w[fms->n-1];
481          wec += web + wea;
482      }
483      wbr = wei * wec;
484
485      /* Add water film. */
486      wbr += fms->wfilm;
487      if ( fms->hf < 0.99 )
488          fms->wfilm = 0.;
489      if ( wbr > fms->wmx )
490          wbr = fms->wmx;
491      return (wbr);
492  }
493
494  /*
495  ***************************************************************************
496  *
497  *  NAME
498  *      Fms_Reconfigure()
499  *
500  *  SYNOPSIS
501  *      #include "fms.h"
502  *      static long Fms_Reconfigure ( Fms *fms );
503  *
504  *  ARGUMENTS
505  *      fms     Pointer to fuel moisture stick object.
506  *
507  *  RETURNS
508  *      Returns FMS_OK on success or FMS_ERROR on failure.
509  *
510  *  DESCRIPTION
511  *      Recalculates all stick elements from current node, radius, density,
512  *      timestep, and barometric pressure values.  Also calculates
513  *      intermediate parameters and optimization factors that depend solely
514  *      on manifest constants, derived constants, and stick properties.
515  ***************************************************************************
516  */
517
518  static long Fms_Reconfigure ( Fms * fms )
519  {
520      size_t i;
521      double rcav;
522
523      /* Free existing nodes. */
524      if ( fms->x ) free(fms->x);
525      if ( fms->d ) free(fms->d);
526      if ( fms->w ) free(fms->w);
527      if ( fms->s ) free(fms->s);
528
```

```
529     /* Create new nodal parameters. */
530     if ( (fms->x = (double *) malloc(fms->n * sizeof(double))) == NULL
531        || (fms->d = (double *) malloc(fms->n * sizeof(double))) == NULL
532        || (fms->s = (double *) malloc(fms->n * sizeof(double))) == NULL
533        || (fms->w = (double *) malloc(fms->n * sizeof(double))) == NULL )
534     {
535         Fms_Destroy(fms);
536         return(FMS_ERROR);
537     }
538
539     /* dx = internodal distance (cm). */
540     fms->dx = fms->a / (fms->n - 1);
541
542     /* rdur = elapsed time since start of the current rainfall episode (hr). */
543     fms->rdur = 0.;
544
545     /* wmax = maximum possible stick moisture content (g/g). */
546     fms->wmax = (1. / fms->dp) - (1 / 1.53);
547
548     /* Stick nodal values. */
549     for (i=0; i<fms->n; i++)
550     {
551         /* x = radial distance from center of stick (cm). */
552         fms->x[i] = fms->a - (fms->dx * i);
553         /* d = bound water diffusivity (cm2/hr). */
554         fms->d[i] = 0.;
555         /* s = saturation (g/g). */
556         fms->s[i] = 0.;
557         /* w = water content (g/g). */
558         fms->w[i] = 1.;
559     }
560
561     /* Initialization flag. */
562     fms->updates = 0;
563     fms->state = FMS_STATE_None;
564     fms->init = 0;
565
566     /*
567      *-----------------------------------------------------------------------
568      *  Stick property-dependent optimization factors.
569      *-----------------------------------------------------------------------
570      */
571
572     /* amlf = aml optimization factor. */
573     fms->amlf = fms->Hwf / (0.24 * fms->bp * fms->dp * fms->a);
574
575     /* capf = cap optimization factor. */
576     rcav = 0.5 * fms->Aw * fms->Wl;
577     fms->capf = 3600. * fms->Pi * fms->St * rcav * rcav
578               / ( 16. * fms->a * fms->a * fms->al * fms->dp );
579
580     /* sf = s factor used in Fms_Update() loop. */
581     fms->sf = 3600. * fms->mdt / (2. * fms->dx * fms->dp);
582
583     /* pro = ???; used only in computation of Pi, the water vapor pressure */
584     /* at stick surface (cal/cm3). */
585     fms->pro = fms->Ap * fms->bp;
586
587     /* Some silly little time savers. */
588     fms->dx_2  = fms->dx * 2.;
589     fms->mdt_2 = fms->mdt * 2.;
590
591     return(FMS_OK);
592 }
593
594 /*-------------------------------------------------------------------------
```

```
595 /*****************************************************************************
596 *
597 *  NAME
598 *      Fms_Rederive()
599 *
600 *  SYNOPSIS
601 *      #include "fms.h"
602 *      Fms *Fms_Rederive( Fms *fms );
603 *
604 *  ARGUMENTS
605 *      fms     Pointer to the Fms object to derive constants.
606 *
607 *  RETURNS
608 *      Returns nothing.
609 *
610 *  DESCRIPTION
611 *      Dervies all the "constants" and optimization factors that rely solely
612 *      on the Fms object's "manifest constants".
613 *
614 *  NOTES
615 *      Manifest constants used by the fuel moisture computations are assigned
616 *      to each object in Fms_CreateCustom().  This permits experimentation and
617 *      comparison between sticks using different "constants".  Derived
618 *      constants and optimization factors based solely upon the manifest
619 *      constants are then calculated here.
620 *
621 *      A program wishing to change or test the constants should:
622 *          (1) Call Fms_CreateCustom(...),
623 *          (2) Directly change any constants for the Fms object, for example:
624 *                  fms->Alb = 0.55
625 *          (3) Call Fms_Rederive(fms)
626 *          (4) Call Fms_Reconfigure(fms)
627 *****************************************************************************
628 */
629
630 void Fms_Rederive ( Fms *fms )
631 {
632     double sb, tsk;
633
634     /*
635      *-------------------------------------------------------------------------
636      * Srf = factor used to derive fms->sr, the solar radiation received by
637      *       half the stick (cal/cm2-hr), such that:
638      *           fms->sr = fms->Srf * fsc
639      *       instead of
640      *           fms->sr = 0.5 * fsc * (1. - fms->Alb) * 116.4
641      *       where "fsc" is solar radiation expressed as a fraction of
642      *       the solar constant.
643      *-------------------------------------------------------------------------
644      */
645
646     fms->Srf = (1.0 - fms->Alb) * 116.4 / fms->Pi;
647
648     /*
649      *-------------------------------------------------------------------------
650      * Wsf = factor to derive fms->wsa, stick fiber saturation point (g/g),
651      *       such that:
652      *           wsa = c1 * wsf ** c2
653      *       instead of:
654      *           wsa = c1 * (-log(1.0 - 0.99)) ** c2
655      *-------------------------------------------------------------------------
656      */
657
658     fms->Wsf = -log(1.0 - 0.99);
659
660     /*
```

```
661     /*-------------------------------------------------------------------
662      * Hrd and Hrn are factors used to optimize the datytime and nighttime
663      * longwave radfative surface heat transfer coefficient (cal/cm2-hr-C).
664      * Either Hrd or Hrn are assigned to "hr", which is used in the stick
665      * temperature calculations
666      *-------------------------------------------------------------------
667      */
668
669     sb = 4. * 3600. * fms->Sbc * fms->Eps;
670
671     tsk = fms->Tcd + fms->Kelvin;
672     fms->Hrd = sb * tsk * tsk * tsk / fms->Pi;
673
674     tsk = fms->Tcn + fms->Kelvin;
675     fms->Hrn = sb * tsk * tsk * tsk / fms->Pi;
676
677     /*
678      *-------------------------------------------------------------------
679      * Sir = value of fms->s[] below which continuous columns of liquid water
680      *       no longer exist.
681      *-------------------------------------------------------------------
682      */
683
684      fms->Sir = fms->Aw * fms->Alpha / (4. * (1.-(2.-fms->Aw) * fms->Alpha));
685
686     /*
687      *-------------------------------------------------------------------
688      * Scr = value of fms->s[] at which a liquid miniscus first enters the
689      *       tapered portion of wood cells.
690      *-------------------------------------------------------------------
691      */
692
693      fms->Scr = 4. * fms->Sir;
694
695     /* Vf = optimization factor used in Fms_Update(). */
696     fms->Vf = fms->St / (fms->Aw * fms->Wl * fms->Scr);
697
698     /* Hwf = hw and aml computation factor used in Fms_Update(). */
699     fms->Hwf = 0.622 * fms->Hc * pow((fms->Pr / fms->Sc), (0.667));
700
701     return;
702 }
703
704 /*
705  ***********************************************************************
706  *
707  *  NAME
708  *      Fms_Update()
709  *
710  *  SYNOPSIS
711  *      #include "fms.h"
712  *      void Fms_Update (Fms *fms, double eHr, double aC, double hFtn,
713  *          double sMv, double rcumCm);
714  *
715  *  ARGUMENTS
716  *      fms     Pointer to fuel moisture stick object to update.
717  *
718  *  RETURNS
719  *      The function returns nothing.
720  *
721  *  DESCRIPTION
722  *      Ralph Nelson's 10-hour stick fuel moisture prediction model.
723  *      Updates the stick's temperature and moisture variables based upon
724  *      changes in the stick's environmental variables since the last update.
725  *
726  *  SIDE EFFECTS
```

```
727 /*****************************************************************************
728 *  Updates the sticks temperature (fms->t), saturation (fms->s[]), and
729 *  moisture (fms->w[]) values.  If the diffusivity time limit is
730 *  exceeded, diffusivity (fms->d) is also updated.
731 *****************************************************************************/
732
733 void Fms_Update (
734     Fms   *fms,                 /* Pointer to the fuel moisture stick object. */
735     double eHr,                 /* Elapsed time since last observation  (hr) */
736     double aC,                  /* New air temperature                   (C) */
737     double hFtn,                /* New air humidity                  (g/g) */
738     double sMv,                 /* New solar radiation           (milliVolts) */
739     double rcumCm               /* New cumulative rainfall               (cm) */
740 )
741 {
742     double ak;          /* permiability of stick when nonsaturated      (cm2) */
743     double aml;         /* factor related to rate of condensation   ((g/g)/hr) */
744     double bi;          /* mass transfer biot number                     (dl) */
745     double c1;          /* Emc sorption isotherm parameter               (g/g) */
746     double c2;          /* Emc sorption isotherm parameter               (g/g) */
747     double ddtNext;     /* next time (dt) to run diffusivity calculations (hr) */
748     double dhdm;        /* Reciprocal slope of the sorption isotherm          */
749     double fsc;         /* solar constant interpolated between obs       (mv) */
750     double gnu;         /* kinematic viscosity of liquid water      (cm2/s) */
751     double ha;          /* air humidity interpolated between obs         (dl) */
752     double hf_log;      /* -log(1. - fms->hf)                            (g/g) */
753     double hr;          /* longwave radiative surface heat transfer(cal/cm2-hr-C) */
754     double hw;          /* stick heat transfer coeff for vapor diffusion above FSP*/
755     double rai;         /* rainfall runoff factor                   ((g/g)/hr) */
756     double pl;          /* water vapor press at the stick surface   (cal/cm3) */
757     double pa;          /* water vapor pressure in air              (cal/cm3) */
758     double par;         /* dummy variable for s                          (dl) */
759     double ps1;         /* water satur vapor press at surface temp  (cal/cm3) */
760     double psa;         /* water satur vapor press in ambient air   (cal/cm3) */
761     double psd;         /* water satur vapor press at dewpoint      (cal/cm3) */
762     double qv;          /* latent heat of vaporization of water     (cal/mol) */
763     double qw;          /* differential heat of sorption of water   (cal/mol) */
764     double sr;          /* solar rad received by half the stick  (cal/(cm2*hr)) */
765     double stca;        /* surface mass transfer for adsorption  ((cm3/cm2)/hr) */
766     double sv;          /* current solar radiation                  (millivolts) */
767     double tfd;         /* intermediate stick surface temperature        (C) */
768     double tfract;      /* fraction of time elapsed between updates      (dl) */
769     double ta;          /* air temperature interpolated between obs      (C) */
770     double tdp;         /* dewpoint temperature                          (C) */
771     double tdw;         /* dewpoint temperature                          (K) */
772     double tka;         /* air temperature                               (K) */
773     double tkf;         /* stick temperature                             (K) */
774     double tsk;         /* sky temperature                               (K) */
775     double tt;          /* elapsed moisture computation time             (hr) */
776     double v;           /* product of changing variables in the DE for new s  */
777     double w1o;         /* previous value of w[0]                        (g/g) */
778     double wdiff;       /* maximum minus current fiber saturation        (g/g) */
779     double g[FMS_MAX_NODES];  /* derivative of w wrt x              ((g/g)/cm) */
780     double o[FMS_MAX_NODES];  /* derivative of s wrt x                   (1/cm) */
781     double p[FMS_MAX_NODES];  /* product of x, g, and d          ((g/g)-cm2/hr) */
782     double u[FMS_MAX_NODES];  /* product of v and o                         (?) */
783     size_t i, less, n, do_gnu;
784
785     /* Increment update counter */
786     fms->updates++;
787
788     /* If stick is not initialized, do so using environmental variables. */
789     if ( ! fms->init )
790         Fms_Initialize(fms, aC, hFtn, sMv, rcumCm, aC, hFtn, 0.55*hFtn);
791
792     /* Initialize automatics. */
```

```
793     for ( i=0; i<FMS_MAX_NODES; i++ )
794         o[i] = g[i] = p[i] = u[i] = 0.0;
795
796     /* Save previous stick values. */
797     fms->ta0 = fms->ta1;          /* Previous air temperature (C). */
798     fms->ha0 = fms->ha1;          /* Previous air relative humidity (g/g). */
799     fms->sv0 = fms->sv1;          /* Previous pyranometer volatgae (mvolts). */
800     fms->f0  = fms->f1;           /* Prev rainfall received by stick (cm/hr). */
801
802     /* Save current environment values. */
803     fms->ta1 = aC;                /* Current air temperature (C). */
804     fms->ha1 = hFtn;              /* Current air relative humidity (g/g). */
805     fms->sv1 = sMv;               /* Current pyranometer voltage (millivolts). */
806     fms->f1 = rcumCm/eHr/fms->Pi; /* Rainfall received by stick (cm/hr). */
807
808     /* Accumulate rainfall event duration, which may be reset below. */
809     fms->rdur += eHr;
810
811     /* ddtNext = next time (hr) to run diffusivity computations. */
812     ddtNext = fms->ddt;
813
814     /* Loop for each moisture time step between environmental inputs. */
815     for (tt = fms->mdt; tt <= eHr + fms->mdt; tt += fms->mdt )
816     {
817         /* Interpolate environmental values between old and current obs. */
818         tfract = tt / eHr;
819         ta = fms->ta0 + (fms->ta1 - fms->ta0) * tfract;
820         ha = fms->ha0 + (fms->ha1 - fms->ha0) * tfract;
821         sv = fms->sv0 + (fms->sv1 - fms->sv0) * tfract;
822
823         /* fsc = fraction of the solar constant. */
824         fsc = 0.07 * sv;
825
826         /* tka = ambient air temperature (K). */
827         tka = ta + fms->Kelvin;
828
829         /* tdw = dew point temperature (K). */
830         tdw = 5205. / ((5205. / tka) - log(ha));
831
832         /* tdp = dew point temperature (C). */
833         tdp = tdw - fms->Kelvin;
834
835         /* Day- and night-time parameters. */
836         /* tsk = sky temperature (K). */
837         /* hr = longwave radiative surface heat transfer coeff (cal/cm2-hr-C). */
838         /* sr = solar radiation received by half the stick (cal/cm2-hr). */
839         if ( fsc < 0.000001 )
840         {
841             tsk = fms->Tcn + fms->Kelvin;
842             hr = fms->Hrn;
843             sr = 0;
844         } else {
845             tsk = fms->Tcd + fms->Kelvin;
846             hr = fms->Hrd;
847             sr = fms->Srf * fsc;
848         }
849
850         /* psa = water saturation vapor pressure in ambient air (cal/cm3). */
851         psa = 0.0000239 * exp(20.58 - (5205. / tka));
852
853         /* pa = water vapor pressure in air (cal/cm3). */
854         pa = ha * psa;
855
856         /*
857          * ------------------------------------------------------------------
858          * Compute stick temperature and surface humidity.
```

```
859             *-------------------------------------------------------------
860             */
861
862             /* tfd = intermediate stick surface temperature (C). */
863             tfd = ta + (sr - hr * (ta - tsk + fms->Kelvin)) / (hr + fms->Hc);
864
865             /* qv = latent heat of vaporization of water (cal/mole). */
866             qv = 13550. - 10.22 * (tfd + fms->Kelvin);
867
868             /* qw = differential heat of sorption of water (cal/mole). */
869             qw = 5040. * exp(-14. * fms->w[0]);
870
871             /* hw = stick heat transfer coeff for vapor diffusion above FSP. */
872             hw = (fms->Hwf * fms->Ap / 0.24) * qv / 18.;
873
874             /* fms->tf = stick temperature (C). */
875             fms->tf = tfd - (hw * (tfd - ta) / (hr + fms->Hc + hw) );
876             tkf = fms->tf + fms->Kelvin;
877
878             /* c1 = EMC sorption isotherm parameter (g/g). */
879             c1 = 0.1617 - 0.001419 * fms->tf;
880
881             /* c2 = EMC sorption isotherm parameter (g/g). */
882             c2 = 0.4657 + 0.003578 * fms->tf;
883
884             /* wsa = stick fiber saturation point (g/g). */
885             fms->wsa = c1 * pow(fms->Wsf, c2);
886             wdiff = fms->wmax - fms->wsa;
887
888             /* ps1 = water saturation vapor pressure at surface temp (cal/cm3). */
889             ps1 = 0.0000239 * exp(20.58 - (5205. / tkf));
890
891             /* p1 = water vapor pressure at stick surface (cal/cm3). */
892             if ( (p1 = pa + fms->pro * (qv / (qv + qw)) * (tka - tkf)) < 0.000001 )
893                 p1 = 0.000001;
894
895             /* fms->hf = stick humidity (g/g). */
896             if ( (fms->hf = p1 / ps1) > 0.99 )
897                 fms->hf = 0.99;
898
899             /*
900              *-------------------------------------------------------------
901              * Compute surface moisture content.
902              *-------------------------------------------------------------
903              */
904             w1o = fms->w[0];
905             aml = 0.;
906             fms->state = FMS_STATE_None;
907
908 /*  . . . If it is raining: */
909             if ( fms->f1 > fms->f0 )
910             {
911                 fms->tf = tfd;
912                 fms->hf = 0.99;
913
914 /*  . . . . . If this is a rainstorm: */
915                 if ( (fms->f1 - fms->f0) >= 0.0365 )
916                 {
917                     fms->state = FMS_STATE_Rainstorm;
918                     fms->w[0] = fms->wmx;
919                     fms->wfilm = 0.0485 * 0.15 / 0.105;
920                     fms->fcum = fms->f1 - fms->f0;
921                 }
922 /*  . . . . . Else this is a gentle rain: */
923                 else
924                 {
```

```
925                 fms->state = FMS_STATE_Rainfall;
926                 fms->wfilm = 0.;
927 /*..... 1b1: if within first hour of gentle rainfall duration. */
928                 if ( fms->rdur <= 1.0 )
929                 {
930                     fms->fcum = fms->f1 - fms->f0;
931                     /* rai = factor related to runoff during 1st hr of rain. */
932                     rai = 0.8 * (1.0 - exp(-100. * fms->fcum));
933                     if ( fms->ha1 < fms->ha0 )
934                         rai *= 0.15;
935                     fms->w[0] += (rai * fms->mdt);
936                 }
937 /*..... 1b2: else after first hour of gentle rainfall duration. */
938                 else
939                 {
940                     fms->fcum += (fms->f1 - fms->f0);
941                     fms->w[0] += (0.07 * fms->fcum * fms->mdt);
942                 }
943
944                 if ( fms->w[0] > fms->wmx )
945                     fms->w[0] = fms->wmx;
946
947                 if ( (fms->s[0] = (fms->w[0] - fms->wsa) / wdiff) < 0. )
948                     fms->s[0] = 0.;
949             }   /* end of gentle rain */
950         }   /* end of rain */
951
952 /*..... 2: else it is not raining. */
953         else
954         {
955             fms->rdur = 0.;
956             fms->fcum = 0.;
957             fms->wfilm = 0.;
958
959 /*..... 2a: if moisture content greater than fiber saturation point. */
960             if ( fms->w[0] > fms->wsa )
961             {
962                 p1 = ps1;
963                 fms->hf = 0.99;
964
965                 /* psd = water saturation vapor pressure at dewpt (cal/cm3). */
966                 psd = 0.0000239 * exp(20.58 - (5205. / tdw));
967
968                 /* aml = factor related to rate of evap or conden.((g/g)/hr). */
969                 if ( fms->tf > tdp || p1 <= psd )
970                     aml = fms->amlf * (ps1 - psd);
971
972                 fms->w[0] = fms->w[0] - aml * fms->mdt_2;
973                 if ( aml > 0. )
974                 {
975                     gnu = 0.00439 + 0.00000177 * pow((338.76-tkf), 2.1237);
976                     fms->w[0] = fms->w[0] - fms->mdt * fms->capf / gnu;
977                 }
978
979                 if ( fms->w[0] > fms->wmx )
980                     fms->w[0] = fms->wmx;
981
982 /*..... 2a1: if moisture content is rising= condensation */
983                 if ( fms->w[0] > w1o )
984                 {
985                     fms->state = FMS_STATE_Condensation;
986                     fms->s[0] = (fms->w[0] - fms->wsa) / wdiff;
987                 }
988
989 /*..... 2a2: else if moisture content is steady: stagnation */
990                 else if ( fms->w[0] == w1o )
```

```
991                 {
992                     fms->state = FMS_STATE_Stagnation;
993                 }
994
995 /*..........2a3: else if moisture content is falling: evaporation */
996                 else if ( fms->w[0] < w1o )
997                 {
998                     fms->state = FMS_STATE_Evaporation;
999                     if ( (fms->s[0] = (fms->w[0] - fms->wsa) / wdiff) < 0. )
1000                        fms->s[0] = 0.;
1001                }
1002            }
1003 /*......2b: else if fuel temperature is less than dewpoint: condensation */
1004            else if ( fms->tf <= tdp )
1005            {
1006                fms->state = FMS_STATE_Condensation;
1007
1008                /* psd = water saturation vapor pressure at dewpt ((cal/cm3)) */
1009                psd = 0.0000239 * exp(20.58 - (5205. / tdw));
1010                if ( p1 <= psd )
1011                    aml = fms->amlf * (p1 - psd);
1012
1013                fms->w[0] = fms->w[0] - aml * fms->mdt_2;
1014                if ( (fms->s[0] = (fms->w[0] - fms->wsa) / wdiff) < 0. )
1015                    fms->s[0] = 0.;
1016            }
1017 /*..........2c: else surface moisture content less than fiber saturation pt */
1018 /*              and stick temperature greater than dewpoint ... */
1019            else
1020            {
1021                /* fms->sem = stick equilibrium moisture content (g/g) */
1022                hf_log = -log(1. - fms->hf);
1023                fms->sem = c1 * pow(hf_log, c2);
1024
1025                if ( fms->w[0] != fms->sem )
1026                {
1027 /*..............2c1: if surface moisture greater than equilibrium: desorption */
1028                    if (fms->w[0] > fms->sem)
1029                    {
1030                        fms->state = FMS_STATE_Desorption;
1031                        bi = fms->Stcd * fms->dx / fms->d[0];
1032                    }
1033 /*.............2c2: else surface moisture less than equilibrium: adsorption */
1034                    else
1035                    {
1036                        fms->state = FMS_STATE_Adsorption;
1037                        /* dhdm = reciprocal slope of the sorption isotherm */
1038                        dhdm = (1. - fms->hf) * pow(hf_log, (1. - c2))
1039                             / (c1 * c2);
1040
1041                        /* stca = surface mass transfer coeff for adsorption */
1042                        /*        ((cm3/cm2)/hr) */
1043                        stca =  fms->Hwf * ps1 * dhdm
1044                             / (0.24 * fms->dp * fms->bp);
1045                        if ( stca < fms->Stcaf )
1046                            stca = fms->Stcaf;
1047                        bi = stca * fms->dx / fms->d[0];
1048                    }
1049                    fms->w[0] = (fms->w[1] + bi * fms->sem) / (1. + bi);
1050                    fms->s[0] = 0.;
1051                }
1052            }
1053        }   /* end of not raining */
1054
1055        /*
1056         *-----------------------------------------------------------------
```

```
1057            * Compute interior nodal moisture content values.
1058            *-----------------------------------------------------------------------
1059            */
1060
1061          for (i=1; i<fms->n-1; i++)
1062          {
1063              /* g = derivative of w with respect to x ((g/g)/cm). */
1064              g[i] = (fms->w[i+1] - fms->w[i-1]) / (fms->dx_2);
1065              /* o = derivative of s with respect to x (1/cm). */
1066              o[i] = (fms->s[i+1] - fms->s[i-1]) / (fms->dx_2);
1067          }
1068          g[0] = (fms->w[1] - fms->w[0]) / fms->dx;
1069          o[0] = (fms->s[1] - fms->s[0]) / fms->dx;
1070          g[fms->n-1] = 0.;
1071          o[fms->n-1] = 0.;
1072
1073          /* Skip the rest if in stagnation state. */
1074          if (fms->state != FMS_STATE_Stagnation)
1075          {
1076              do_gnu = 1;      /* do gnu computation once per mdt iteration */
1077
1078              for (i=0; i<fms->n; i++)
1079              {
1080                  v = 0.;
1081                  fms->s[i] = 0.0;
1082                  par = (fms->w[i] - fms->wsa) / wdiff;
1083                  if (par > 0.)
1084                  {
1085                      fms->s[i] = par;
1086                      if ( par > fms->Sir && par < fms->Scr )
1087                      {
1088                          /* ak = permeability of stick under nonsaturated. */
1089                          /*      conditions (cm2). */
1090                          ak = fms->Aks * (2. * sqrt(par / fms->Scr) - 1.);
1091
1092                          /* gnu = kinematic viscosity of liquid water (cm2/s). */
1093                          if ( do_gnu == 1 )  /* do this just one time */
1094                          {
1095                              gnu = 0.00439 + 0.00000177
1096                                  * pow((338.76 - tkf), 2.1237);
1097                              do_gnu = 0;      /* don't do this again */
1098                          }
1099
1100                          /* v = product of changing variables in the D,E. */
1101                          /*     for calculating a new "s". */
1102                          v = (ak / gnu) * fms->x[i] * fms->Vf
1103                            * pow((fms->Scr / par), 1.5);
1104                      }
1105                  }   /* end if par > 0. */
1106                  u[i] = v * o[i];
1107              }
1108
1109              for (i=1; i<fms->n-1; i++)
1110              {
1111                  fms->s[i] += fms->sf * (u[i+1] - u[i-1]) / (fms->x[i] * wdiff);
1112                  if (fms->s[i] > 1.)
1113                      fms->s[i] = 1.;
1114              }
1115              fms->s[fms->n-1] = fms->s[fms->n-2];
1116
1117              /* Check if fms->s[] is less than fms->Sir (limit of */
1118              /* continuous liquid columns) at ANY stick node. */
1119              for (less=0, i=1; i<fms->n-1; i++)
1120              {
1121                  if (fms->s[i] < fms->Sir )
1122                  {
```

```
1123                        less = 1;
1124                        break;
1125                    }
1126                }
1127
1128                /* If all nodes have continuous liquid columns (s >= Sir): */
1129                if ( less == 0 )
1130                {
1131                    for (i=1; i<fms->n-1; i++)
1132                    {
1133                        if ( (fms->w[i] = fms->wsa + fms->s[i] * wdiff) > fms->wmx )
1134                            fms->w[i] = fms->wmx;
1135                    }
1136                }
1137                /* Else at least one node has s < Sir: */
1138                else
1139                {
1140                    for (i=1; i<fms->n-1; i++)
1141                    {
1142                        n = i + 1;
1143                        p[n] = g[n] * fms->x[n] * fms->d[n];
1144                        p[i] = g[i] * fms->x[i] * fms->d[i];
1145                        fms->w[i] += fms->mdt * (p[n] - p[i])
1146                                  / (fms->dx * fms->x[i]);
1147                        if ( fms->w[i] > fms->wmx )
1148                            fms->w[i] = fms->wmx;
1149                    }
1150                }
1151                fms->w[fms->n-1] = fms->w[fms->n-2];
1152            }
1153
1154            /* Update moisture diffusivity if within less than half a time step. */
1155            if ( (ddtNext - tt) < (0.5 * fms->mdt) )
1156            {
1157                Fms_Diffusivity(fms);
1158                ddtNext += fms->ddt;
1159            }
1160        }   /* Next observation */
1161
1162        return;
1163    }
1164
1165    /*
1166     ************************************************************************
1167     *   End of fms.c
1168     ************************************************************************
1169     */
```

Appendix C

fmstest Source Code File
fmstest.c

```
/*
*******************************************************************************
*
*  NAME
*      fmstest.c
*
*  AUTHOR
*      Copyright (c) 1997 by Collin D. Bevins.
*      See the file "license.terms" for information on usage and
*      redistribution of this file, and for a DISCLAIMER OF ALL WARRANTIES.
*
*  DESCRIPTION
*      Simple driver program for the Fuel Moisture Stick (FMS) C library.
*      This program reads weather data from standard input,
*      determines the mean temperature and mean and surface moisture contents
*      for a standard 10-hour fuel stick, and writes the results to standard
*      output.
*
*  USAGE
*      "fmstest" is a filter program that is run by invoking its name with
*      zero or more option-value arguments:
*
*      fmstest [<-option> <value> ...]
*
*      The following argument option-value pairs define initial fuel stick
*      environment:
*
*          Option      Units   Description             Default
*          ------      -----   --------------------    ---------------------
*          -hair       g/g     initial air humidity    [0.80 g/g]
*          -hfuel      g/g     initial fuel humidity   [0.80 g/g]
*          -moisture   g/g     initial fuel moisture   [0.20 g/g]
*          -rain       cm      initial cumulative rain [0.0 cm]
*          -solar      w/m2    initial solar radiation [0 watts/m2]
*          -tair       C       initial air temperature [10.0 C]
*          -tfuel      C       initial fuel temperate  [10.0 C]
*
*
*      The following argument option-value pairs define custom fuel stick
*      properties:
*
*          Option      Units   Description             Default
*          ------      -----   --------------------    ---------------------
*          -ddt        hr      diffusivity time step   [0.25 hours]
*          -density    g/cm3   stick density           [0.40 g/cm3]
*          -length     cm      stick length            [50.0 cm]
*          -mdt        hr      moisture time step      [1/60 hours]
*          -nodes      n       computation nodes       [21 nodes]
*          -pressure   cal/cm3 barometric pressure     [0.0218 cal/cm3]
*          -radius     cm      stick radius            [0.64 cm]
*          -wmax       g/g     max moisture content    [0.60 g/g]
*
*
*      The following argument option-value pairs modify i/o defaults:
*
*          Option      Units   Description             Default
*          ------      -----   --------------------    ---------------------
*          -annotate   (y|m)   add annotation          [no]
*          -delimit    (space | comma)                 [space]
*          -input      file    input file name         [standard input]
*          -output     file    output file name        [standard output]
*          -verbose    (y|n)   verbose output          [no]
*
*      NOTES
*          1   All options must be preceeded by a hyphen.
*
```

```
67  *          2   All options must be lower case.
68  *
69  *          3   Options may be abbreviated to their shortest unique prefix
70  *              (-a, -dd, -del, -den, -h, -i, -l, -md, -mo, -n, -o, -p, -rad,
71  *              -rai, -t, -v, or -w).
72  *
73  *          4   If -tfuel is not specified, the initial stick temperature is
74  *              the air temperature from the first weather observation.
75  *
76  *          5   If -hfuel is not specified, the initial stick surface
77  *              relative humidity is the air relative humidity from the first
78  *              weather observation.
79  *
80  *          6   If -moisture is not specified, the initial stick moisture
81  *              content is calculated from the first weather observation as:
82  *                  moisture = .55 * RH/100.
83  *
84  *          7   If -input is not specified, input is read from "standard
85  *              input", which is the keyboard.  This is probably not
86  *              what you want, but in fact allows fmstest to operate as
87  *              a pipe between two other programs.  To use an input file,
88  *              use the -input option OR the input redirection (via <).
89  *
90  *          8   If -output is not specified, output is written to "standard
91  *              output", which is the console.  It may also be piped into
92  *              a subsequent program (such as "cut" or "paste") or
93  *              redirected to a file using output redirection (via >), or
94  *              by using the -output option.
95  *
96  *  INPUT FILE FORMAT
97  *      Input is a series of two or more weather records.
98  *      Each weather record contains at least five space-delimited fields:
99  *
100 *      Field   Variable                            Units
101 *      -----   ----------------------------------  --------
102 *        1     Elapsed time since previous obs     hours
103 *        2     Air temperature at stick surface    C
104 *        3     Relative humidity at stick surface  percent
105 *        4     Solar radiation at stick surface    watts/m2
106 *        5     Cumulative rainfall                 mm
107 *      6-10    Optional annotations                text
108 *
109 *      Notes:
110 *          1   All fields must be separated by one or more spaces or tabs.
111 *          2   Elapsed time for the first record is ignored.
112 *          3   Cumulative rainfall for the first record should be zero.
113 *          4   Up to 5 additional space-delimited strings may be added
114 *              to the end of each record.  The most likely use of these
115 *              strings include date/time stamps or field notes.
116 *          5   Input records may not exceed 1023 characters.
117 *          6   Each text field may not exceed 127 characters.
118 *          7   Any record beginning with a pound #, asterisk *, or bang !
119 *              is considered a comment line, and if -verbose is on, is
120 *              echoed to the output.
121 *
122 *  OUTPUT FILE FORMAT
123 *      Output consists of a single record for every weather observation.
124 *      The number and order of fields in the output depends upon the
125 *      -annotate and -verbose options.
126 *
127 *      The first three output fields are always:
128 *      Field   Variable (units)
129 *      -----   ------------------------------------
130 *        1     Stick temperature (C)
131 *        2     Stick mean moisture content (g/g)
132 *        3     Stick surface moisture content (g/g)
```

```
133 *
134 *       If -verbose is yes, the next five fields are:
135 *       Field   Variable (units)
136 *       -----   ------------------------------------
137 *         4     Elapsed time since previous obs (hr)
138 *         5     Air temperature at stick surface (C)
139 *         6     Relative humidity at stick surface (%)
140 *         7     Solar radiation at stick surface (watts/m2)
141 *         8     Cumulative rainfall (cm)
142 *
143 *       Finally, if -annotate is yes, all annotations are appended to the
144 *       output record.
145 *
146 *  COMPILATION
147 *       cc -g fmstest.c fms.c -lm -o fmstest
148 *
149 *  HISTORY
150 *       Version 1.0
151 *-------------------------------------------------------------------------
152 */
153
154 #include <stdio.h>
155 #include <stdarg.h>
156 #include <string.h>
157 #include "fms.h"
158
159 /*-------------------------------------------------------------------------
160 * Compiler-specific options.
161 *-------------------------------------------------------------------------*/
162
163 #ifdef __TURBOC__
164 extern stklen = 32000;
165 #endif
166
167 /*-------------------------------------------------------------------------
168 * Command line options.
169 *-------------------------------------------------------------------------*/
170
171 static size_t Annotate   = 0;
172 static size_t Comma      = 0;
173 static double Ddt        = 0.25;
174 static double Density    = 0.40;
175 static double Hfuel      = 0.80;
176 static char  *inputFile  = NULL;
177 static double Length     = 50.;
178 static double Mdt        = -1.;
179 static double Moisture   = .20;
180 static size_t Nodes      = 21;
181 static char  *outputFile = NULL;
182 static double Pressure   = 0.0218;
183 static double Radius     = 0.64;
184 static double Tfuel      = 10.;
185 static size_t Verbose    = 0;
186 static double Wmax       = 0.6;
187
188 /*-------------------------------------------------------------------------
189 * Weather observation fields.
190 *-------------------------------------------------------------------------*/
191
192 static double Hair   = 0.80;
193 static double Hours  = 1.0;
194 static double Rain   = 0.;
195 static double Solar  = 0.;
196 static double Tair   = 10.;
197
198 #define MAX_TEXT    128
```

```
199 static char S0[MAX_TEXT] = "";
200 static char S1[MAX_TEXT] = "";
201 static char S2[MAX_TEXT] = "";
202 static char S3[MAX_TEXT] = "";
203 static char S4[MAX_TEXT] = "";
204
205 /*-----------------------------------------------------------------------------
206  * Input/output file names and streams.
207  *---------------------------------------------------------------------------*/
208
209 static FILE *Fin  = stdin;
210 static FILE *Fout = stdout;
211 static FILE *Ferr = stderr;
212
213 /*-----------------------------------------------------------------------------
214  * Static function prototypes.
215  *---------------------------------------------------------------------------*/
216
217 static int  Error ( int code, const char *format, ... ) ;
218 static int  Fmstest_Options ( int argc, char **argv ) ;
219 static int  Fmstest_Read ( void ) ;
220 static void Fmstest_Write ( Fms *fms ) ;
221
222 /*
223  ******************************************************************************
224  *
225  *  NAME
226  *      main()
227  *
228  *  SYNOPSIS
229  *      #include "fms.h"
230  *      int main ( int argc, char **argv ) ;
231  *
232  *  DESCRIPTION
233  *      Reads fuel moisture weather observations and writes 10-hour stick fuel
234  *      moisture predictions.
235  *
236  *  RETURNS
237  *      Exit status 0 on success, exit status 1 on failure.
238  ******************************************************************************
239  */
240
241 int main ( int argc, char **argv )
242 {
243     Fms *fms;
244     int result;
245
246     /* Show the version. */
247     fprintf(Ferr, "fmstest (v1.0 %s)\n", __DATE__);
248
249     /* Parse the command line options. */
250     if ( Fmstest_Options(argc, argv) != 0 )
251         return (1);
252
253     /* If no moisture time step was specified, calculate the default. */
254     if ( Mdt < 1.e-6 )
255         Mdt = 1. / 60.;
256
257     /* Create the requested fuel moisture stick. */
258     if ( (fms = Fms_CreateCustom("fmstest", Nodes, Radius, Length, Density,
259             Mdt, Ddt, Pressure, Wmax)) == NULL )
260         return (Error(1, "Unable to create the stick instance."));
261
262     /* Initialize the stick. */
263     Fms_Initialize(fms, Tair, Hair, Solar, Rain, Tfuel, Hfuel, Moisture);
264
```

```
265     /* Loop for every weather observation */
266     while ( (result = Fmstest_Read()) > 0 )
267     {
268         Fms_Update(fms, Hours, Tair, Hair, Solar, Rain);
269         Fmstest_Write(fms);
270     }
271
272     /* Clean up and return exit status code. */
273     Fms_Destroy(fms);
274     return (result<0) ? 1 : 0;
275 }
276
277 /*
278  ***************************************************************************
279  *
280  *  NAME:
281  *      Fmstest_Options()
282  *
283  *  DESCRIPTION:
284  *      Parses, validates, and stores command line options.
285  *
286  *  RETURNS:
287  *      0 on success.
288  *      -1 on error.
289  ***************************************************************************
290  */
291
292 static int Fmstest_Options ( int argc, char **argv )
293 {
294     size_t i;
295
296     /* Must be an even number of command line options. */
297     if ( ! argc%2 )
298         return (Error(1,
299             "Uneven number of \"-option value\" pairs in command line arguments.
 -2 "));
300
301     /* Process each -option value pair. */
302     for ( i=1; i<argc; i++ )
303     {
304         /* -annotate */
305         if (      strncmp(argv[i], "-a",  2) == 0 )
306         {
307             i++;
308             if ( argv[i][0] == 'y' )
309                 Annotate = 1;
310             else if ( argv[i][0] == 'n' )
311                 Annotate = 0;
312             else
313                 return (Error(-1,
314                     "\"%s\" value muste be \"yes\" or \"no\"", argv[i-1]));
315         }
316         /* -ddt */
317         else if ( strncmp(argv[i], "-dd", 3) == 0 )
318         {
319             i++;
320             if ( (Ddt = atof(argv[i])) < 0.0000001)
321                 return (Error(-1,
322                     "\"%s\" value must be non-zero positive", argv[i-1]));
323         }
324         /* -delimit */
325         else if ( strncmp(argv[i], "-del", 4) == 0 )
326         {
327             i++;
328             if ( argv[i][0] == 's' )
329                 Comma = 0;
```

```
330             else if ( argv[i][0] == 'c' )
331                 Comma = 1;
332             else
333                 return (Error(-1,
334                     "\"%s\" argument must be \"comma\" or \"space\"",
335                     argv[i-1]));
336         }
337         /* -density */
338         else if ( strncmp(argv[i], "-den", 4) == 0 )
339         {
340             i++;
341             if ( (Density = atof(argv[i])) < 0.0000001)
342                 return (Error(-1,
343                     "\"%s\" value must be non-zero positive", argv[i-1]));
344         }
345         /* -hair */
346         else if ( strncmp(argv[i], "-ha",  3) == 0 )
347         {
348             i++;
349             Hair = atof(argv[i]);
350             if ( Hair < 0.01 || Hair > 0.99 )
351                 return (Error(-1,
352                     "\"%s\" value must be in range [.01 - .99]", argv[i-1]));
353         }
354         /* -hfuel */
355         else if ( strncmp(argv[i], "-hf",  3) == 0 )
356         {
357             i++;
358             Hfuel = atof(argv[i]);
359             if ( Hfuel < 0.01 || Hfuel > 0.99 )
360                 return (Error(-1,
361                     "\"%s\" value must be in range [.01 - .99]", argv[i-1]));
362         }
363         /* -input */
364         else if ( strncmp(argv[i], "-i",  2) == 0 )
365         {
366             i++;
367             inputFile = argv[i];
368             if ( (Fin = fopen(argv[i], "r")) == NULL )
369                 return (Error(-1,
370                     "Unable to open input file \"%s\".", inputFile));
371         }
372         /* -length */
373         else if ( strncmp(argv[i], "-l",  2) == 0 )
374         {
375             i++;
376             if ( (Length = atof(argv[i])) < 0.0000001)
377                 return (Error(-1,
378                     "\"%s\" value must be non-zero positive", argv[i-1]));
379         }
380         /* -mdt */
381         else if ( strncmp(argv[i], "-md", 3) == 0 )
382         {
383             i++;
384             if ( (Mdt = atof(argv[i])) < 0.0000001)
385                 return (Error(-1,
386                     "\"%s\" value must be non-zero positive", argv[i-1]));
387         }
388         /* -moisture */
389         else if ( strncmp(argv[i], "-mo", 3) == 0 )
390         {
391             i++;
392             if ( (Moisture = atof(argv[i])) < 0.0000001)
393                 return (Error(-1,
394                     "\"%s\" value must be non-zero positive", argv[i-1]));
395         }
```

```
396         /* -nodes */
397         else if ( strncmp(argv[i], "-n",  2) == 0 )
398         {
399             i++;
400             if ( (Nodes = atoi(argv[i])) < 2)
401                 return (Error(-1,
402                     "\"%s\" value must be greater than 2.", argv[i-1]));
403         }
404         /* -output */
405         else if ( strncmp(argv[i], "-o",  2) == 0 )
406         {
407             i++;
408             outputFile = argv[i];
409             if ( (Fout = fopen(argv[i], "w")) == NULL )
410                 return (Error(-1,
411                     "Unable to open output file \"%s\".", outputFile));
412         }
413         /* -pressure */
414         else if ( strncmp(argv[i], "-p",  2) == 0 )
415         {
416             i++;
417             if ( (Pressure = atof(argv[i])) < 0.0000001)
418                 return (Error(-1,
419                     "\"%s\" value must be non-zero positive", argv[i-1]));
420         }
421         /* -radius */
422         else if ( strncmp(argv[i], "-rad",  4) == 0 )
423         {
424             i++;
425             if ( (Radius = atof(argv[i])) < 0.0000001)
426                 return (Error(-1,
427                     "\"%s\" value must be non-zero positive", argv[i-1]));
428         }
429         /* -rain */
430         else if ( strncmp(argv[i], "-rai",  4) == 0 )
431         {
432             i++;
433             if ( (Rain = atof(argv[i])) < 0.0)
434                 return (Error(-1,
435                     "\"%s\" value must be non-zero positive", argv[i-1]));
436         }
437         /* -s */
438         else if ( strncmp(argv[i], "-s",  2) == 0 )
439         {
440             i++;
441             Solar = atof(argv[i]);
442             if ( Solar < 0. || Solar > 2000. )
443                 return (Error(-1,
444                     "\"%s\" value must be in range [0 - 2000]", argv[i-1]));
445         }
446         /* -tair */
447         else if ( strncmp(argv[i], "-ta",  3) == 0 )
448         {
449             i++;
450             Tair = atof(argv[i]);
451             if ( Tair < -100. || Tair > 99. )
452                 return (Error(-1,
453                     "\"%s\" value must be in range [-100 - 99]", argv[i-1]));
454         }
455         /* -tfuel */
456         else if ( strncmp(argv[i], "-tf",  3) == 0 )
457         {
458             i++;
459             Tfuel = atof(argv[i]);
460             if ( Tfuel < -100. || Tfuel > 99. )
461                 return (Error(-1,
```

```
462                     "\"%s\" value must be in range [-100 - 99]", argv[i-1]));
463         }
464         /* -verbose */
465         else if ( strncmp(argv[i], "-v",  2) == 0 )
466         {
467             i++;
468             if ( argv[i][0] == 'y' )
469                 Verbose = 1;
470             else if ( argv[i][0] == 'n' )
471                 Verbose = 0;
472             else
473                 return (Error(-1,
474                     "\"%s\" value must be \"yes\" or \"no\"", argv[i-1]));
475         }
476         /* -wmax */
477         else if ( strncmp(argv[i], "-w", 2) == 0 )
478         {
479             i++;
480             if ( (Wmax = atof(argv[i])) < 0.0000001)
481                 return (Error(-1,
482                     "\"%s\" value must be non-zero positive", argv[i-1]));
483         }
484         /* Unknow option */
485         else
486             return (Error(-1, "Unknown command line option \"%s\"", argv[i]));
487     }
488
489     return (0);
490 }
491
492 /*
493  ***************************************************************************
494  *
495  *   NAME
496  *       Fmstest_Read()
497  *
498  *   DESCRIPTION
499  *       Reads and converts the next weather input record.
500  *
501  *   RETURNS
502  *       1 if another record was read ok.
503  *       0 if EOF.
504  *      -1 on error.
505  ***************************************************************************
506  */
507
508 static int Fmstest_Read ( void )
509 {
510     char buffer[1024];
511     static long Line = 0;
512
513     /* Keep reading records until we get a weather observation. */
514     while (++Line)
515     {
516         if ( fgets(buffer, sizeof(buffer), Fin) == NULL )
517         {
518             /* If end-of-file, return 0 to indicate EOF. */
519             if ( feof(Fin) )
520                 return (0);
521             /* If not at end-of-file, then a read error occured. */
522              return (Error(-1,
523                 "Unable to read observation %ld from daily input.", Line));
524         }
525
526         /* Echo record if it is a comment line. */
527         if ( buffer[0] == '#' || buffer[0] == '!' || buffer[0] == '*' )
```

```
528         {
529             if ( Verbose )
530                 fprintf(Fout, "%s", buffer);
531         }
532         /* Otherwise, this is a weather record, so process it. */
533         else
534         {
535             break;
536         }
537     }
538
539     /* Scan the required and any optional fields from the record. */
540     S0[0] = S1[0] =S2[0] = S3[0] = S4[0] = '\0';
541     if ( sscanf(buffer, "%lf %lf %lf %lf %lf %s %s %s %s %s",
542             &Hours, &Tair, &Hair, &Solar, &Rain, S0, S1, S2, S3, S4) < 5 )
543         return (Error(-1,
544             "Unable to scan 5 fields from observation %ld.", Line));
545
546     /* Convert solar value from watts/m2 into millivolts. */
547     Solar /= 94.743;
548
549     /* Convert relative humidity from percent to fraction. */
550     if ( (Hair *= .01) > 0.99 )
551         Hair = 0.99;
552
553     /* Read this record ok, so return 1. */
554     return (1);
555 }
556
557 /*
558  ************************************************************************
559  *
560  *  NAME
561  *      Fmstest_Write()
562  *
563  *  DESCRIPTION
564  *      Writes next fuel moisture stick output record.
565  *
566  *  RETURNS
567  *      Function returns nothing.
568  ************************************************************************
569  */
570
571 static const char *State[] = { "N", "A", "D", "C", "E", "R", "X", "S", "Z"};
572 static void Fmstest_Write ( Fms *fms )
573 {
574     double obsmc;
575     long time;
576
577     obsmc = atof(S2);
578     time = atoi(S1);
579
580     fprintf(Fout, "%4ld  %s  %4ld  %s  %4.1f  %5.3f  %5.3f  %5.3f\n",
581         fms->updates, S0, time, State[fms->state], fms->tf, fms->w[0],
582         Fms_MeanMoisture(fms), obsmc);
583     return;
584
585     if ( Comma )
586         fprintf(Fout, "%2.1f,%5.4f,%5.4f",
587             fms->tf, Fms_MeanMoisture(fms), fms->w[0]);
588     else
589         fprintf(Fout, "%4.1f %5.4f %5.4f",
590             fms->tf, Fms_MeanMoisture(fms), fms->w[0]);
591
592     if ( Verbose )
593     {
```

```
594         if ( Comma )
595             fprintf(Fout, ",%4.3f,%2.1f,%2.1f,%5.4f,%2.1f",
596                     Hours, Tair, 100.*Hair, 94.743*Solar, Rain);
597         else
598             fprintf(Fout, " %7.3f %4.1f %5.1f %5.4f %5.1f",
599                     Hours, Tair, 100.*Hair, 94.743*Solar, Rain);
600     }
601
602     if ( Annotate )
603     {
604         if ( Comma )
605             fprintf(Fout, ",%s,%s,%s,%s,%s", S0, S1, S2, S3, S4);
606         else
607             fprintf(Fout, " %s %s %s %s %s", S0, S1, S2, S3, S4);
608     }
609     fprintf(Fout, "\n");
610
611     return;
612 }
613
614 /*
615 ***************************************************************************
616 *
617 *  NAME
618 *      Error()
619 *
620 *  DESCRIPTION
621 *      Error handling function; prints the error message and returns the code.
622 *
623 *  RETURNS
624 *      Returns the error code passed in the first argument.
625 ***************************************************************************
626 */
627
628 static int Error ( int code, const char *format, ... )
629 {
630     va_list vptr;
631     char buffer[1024];
632
633     va_start(vptr, format);
634     vsprintf(buffer, format, vptr);
635     fprintf(Ferr, "\n\n*** %s\n\n", buffer);
636     va_end(vptr);
637     return(code);
638 }
639
640 /*
641 ***************************************************************************
642 *  End of fmstest.h
643 ***************************************************************************
644 */
645
```